Diseño de Algoritmos. Pseudocódigo

Indice

Unidad 1: Algoritmos y programas

Unidad 2: Resolución de problemas. PseudoCódigo

Crea una carpeta llamada **Programación**, dentro de ella una carpeta llamada **Pseudocódigo** y dentro de ella una carpeta llamada **Actividades.**

Unidad 1: Algoritmos y programas

1. Introducción

Las personas aprenden lenguajes y técnicas de programación porque quieren utilizar el ordenador como herramienta para resolver problemas.

La *resolución de un problema*, mediante el ordenador, se lleva a cabo a través de los siguientes **pasos**:

- **Definición o Análisis del problema.** Se plantea el problema a resolver utilizando metodologías de análisis; lo que se denomina "ciclo de vida" de desarrollo del software.
- **Diseño del *algoritmo*** describe la secuencia *ordenada* de pasos que conducen a la resolución del problema.
- **Transformación de ese algoritmo en un *programa*** mediante el uso de un lenguaje de programación.
- **Ejecución y validación del programa.**

Nosotros estudiaremos a partir del segundo paso: *Diseño del algoritmo*.

2. Concepto de Algoritmo y programa

Un **Algoritmo**, se puede definir como una *fórmula* para resolver un problema. Esa fórmula está expresada como una **secuencia de instrucciones (pasos)** (operaciones de asignación, lectura, escritura, estructuras de control, etc...), en orden, para resolver el problema.

La definición de un algoritmo debe describir **tres partes:**

- Entrada
- Proceso
- Salida

Ejemplo 1. Receta de cocina

Entrada: Ingredientes y utensilios empleados
Proceso: Elaboración de la receta en la cocina
Salida: Terminación del plato

Ejemplo 2:

Un cliente realiza un pedido a un proveedor. El proveedor examina en su banco de datos la ficha de dicho cliente; si el cliente es solvente, entonces el

proveedor acepta el pedido, en caso contrario rechaza el pedido. Redacta el algoritmo correspondiente.

Solución del algoritmo por ***pasos***:

1. Inicio
2. Leer el pedido
3. Examinar la ficha del cliente
4. Si el cliente es solvente aceptar el pedido, en caso contrario rechazar el pedido
5. Fin

Las **características de un algoritmo** son las siguientes:

- Son **independientes** del lenguaje de programación en que se expresan y del ordenador que lo ejecuta.
- Deben ser **precisos**, el orden de realización de cada paso.
- Deben estar bien **definidos**, si se sigue el algoritmo varias veces, el resultado debe ser el mismo cada vez.
- Deben ser **finitos**, debe terminar en algún momento; o sea, deben tener un número finito de pasos.

El **algoritmo** ha de expresarse, si el procesador es el ordenador, en forma de **programa**.

Un **programa** es una secuencia de instrucciones dispuestas una a continuación de otras, escritas en un lenguaje de programación.

Cada **paso** en el algoritmo está expresado por medio de una **instrucción** en el programa.

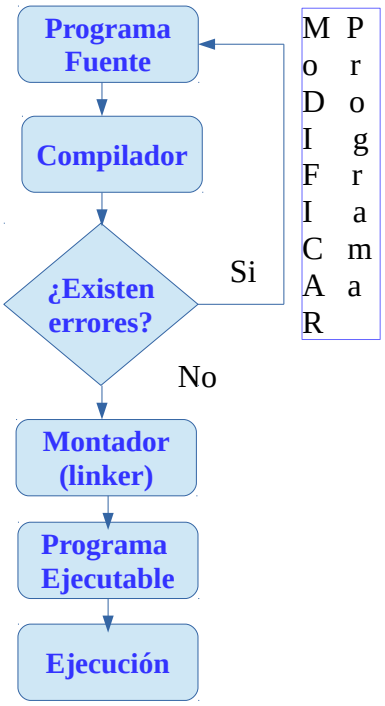
3. Lenguajes de programación

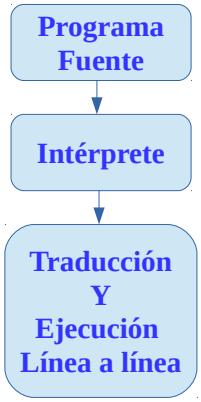
Está claro que para que el ordenador (el procesador) realice un proceso, es necesario que se le suministre un algoritmo adecuado expresado en forma de programa. Y este programa ha de ser, ESCRITO en un lenguaje (idioma) asequible para el humano y TRADUCIDO al lenguaje que entienda el ordenador, el *lenguaje o código máquina* (constituido por **0** y **1**). Esta actividad se denomina **programación**.

Como el *lenguaje máquina* es un código " bajo nivel" que depende del hardware de la máquina, la tarea de programar se hace compleja y tediosa; por esta razón, surgen otros lenguajes llamados de "alto nivel" que ya son cercanos al lenguaje humano.

Para "Traducir" los programas escritos en lenguajes de "alto nivel" (**Programa Fuente**) a lenguaje máquina (**Programa ejecutable**) se precisan de unos programas, los **Traductores**: ***Compiladores*** *o* ***Intérpretes***.

El **compilador** "traduce" el programa fuente a un **programa objeto** (intermedio, en ensamblador) lo que requiere de un proceso de montaje utilizando un programa ***montador o enlazador (linker)*** para convertirlo en programa ejecutable.

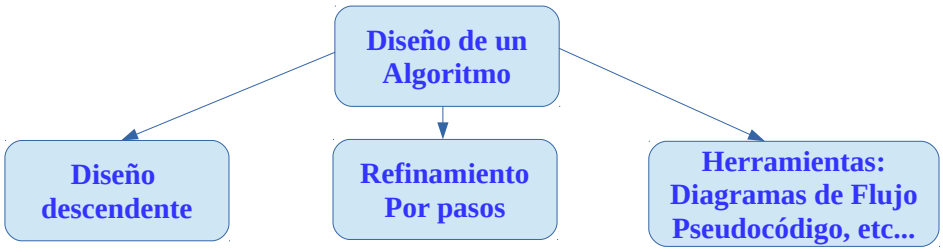




El **intérprete** "traduce" el programa fuente y a continuación lo ejecuta línea a línea. No crea un programa ejecutable. Si el intérprete encuentra un error, el programa se detiene en ese punto. Las instrucciones anteriores se habrían ejecutado.

Podemos decir que el **intérprete** funciona antes que el compilador, porque no tiene que crear el programa ejecutable. Sin embargo, el **compilador** es más rápido cuando se trata de ejecutar cálculos complejos y es independiente de la máquina.

4. Diseño de algoritmos



En la fase del ***diseño de un algoritmo***, para resolver de forma eficaz un problema complejo, lo que se hace es descomponer el problema en *subproblemas* más fáciles de resolver que el original. Este método se denomina ***"Divide y Vencerás"***.

La descomposición de un problema complejo, por el método de "Divide y Vencerás", en distintas subproblemas y éstos a su vez en otros subproblemas más sencillos, se denomina ***"Diseño Descendente"***.

A continuación, se lleva a cabo una descripción más detallada de los ***pasos*** concretos de cada subproblema simple o módulo. Este proceso se denomina, ***Refinamiento del Algoritmo.***

Finalmente, se utilizará la herramienta de programación correspondiente para diseñarlo y representarlo gráficamente. Que puede ser: un diagrama de flujo, pseudocódigo, etc…

Ejemplo: Leer el radio de un círculo y calcular e imprimir su superficie y su circunferencia.

Este problema se puede descomponer o dividir en 3 subproblemas más sencillos:

1. LEER datos de ENTRADA (RADIO)
2. CALCULAR la SUPERFICIE y LONGITUD
3. ESCRIBIR el resultado

Diseño descendente:

Leer Radio
Calcular Superficie
Calcular Longitud
Escribir resultados

Refinamiento:

Leer Radio
Superficie=PI * Radio ^ 2
Longitud = 2 * PI * Radio
Escribir Radio, Longitud, Superficie

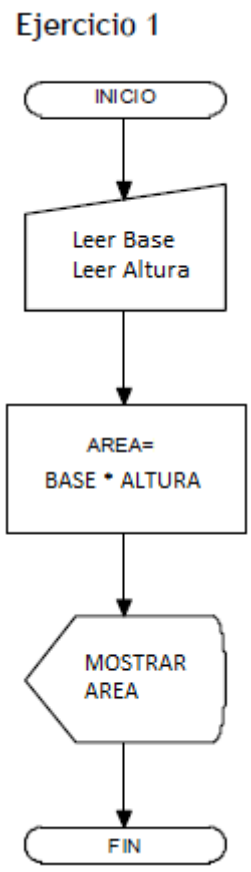
5. Diagramas de flujo

Un diagrama de flujo es una de las técnicas o herramientas de representación de algoritmos más antigua. Se trata de una representación gráfica de las acciones y el orden en que deben realizarse, una tarea o proceso. Comprenden dos grandes bloques: Organigramas y Ordinogramas.

Éstos últimos, los **ordinogramas**, son la representación gráfica del algoritmo. Debe ser amplia y ordenada y no contener errores. En el ordinograma vendrá reflejado: El principio del proceso, las operaciones que se van a realizar, en el orden en el que se han establecido, y el final del proceso.

Ejemplo 1. Calcular el área de un rectángulo cuyos datos base y altura se leen del teclado. Visualizar el resultado en pantalla.

Solución:



Unidad 2: Resolución de problemas. Pseudocódigo

Como ya sabemos el ordenador es una herramienta que se utiliza para la resolución de problemas.

Cuando se plantea un problema, éste ha de resolverse siguiendo las fases que ya conocemos. En la fase del diseño de un algoritmo (especificación formal del algoritmo), podemos utilizar además del diagrama de flujo (ordinograma), otra herramienta para diseñar el algoritmo es, el **Pseudocódigo**.

El **Pseudocódigo** es un lenguaje de especificación de algoritmos, informal con descripciones del lenguaje natural humano. No dispone de sintaxis estándar. Omite declaraciones de variables. Es más fácil de entender que un lenguaje de programación. Y facilita el paso a la codificación en un lenguaje de programación.

Sus ***características***:

- Sintaxis sencilla
- Manejo de estructuras de datos básicas de control
- Cuatro tipos de datos básicos: numéricos, lógicos, carácter y cadenas
- Estructuras de datos: arreglos o arrays

Existe aplicaciones que facilitan la creación de algoritmos utilizando *Pseudocódigo*. Nosotros utilizaremos la herramienta **PseInt** (su nombre proviene de la unión de los vocablos *Pseudocódigo* e *Intérprete*) para escribir los algoritmos y ejecutarlos en nuestro equipo.

En este enlace puedes descargar el instalador de dicha herramienta para tu sistema.
(la URL es: http://pseint.sourceforge.net/index.php?page=portada.php)

En esa página web oficial de **PseInt** podrás encontrar documentación, ejemplos, foros, etc... sobre dicha herramienta y Pseudocódigo.

1. Estructura de un algoritmo en Pseudocódigo

Todo algoritmo en *Pseudocódigo* tiene la siguiente **estructura general**:

Algoritmo Nombre del algoritmo
Declaraciones
<declaraciones de variables y contantes>

Inicio

Secuencia de instrucciones (operaciones de asignación, lectura, escritura, estructuras de control, etc...)

Fin

Ejecuta PseInt, e introduce y ejecuta el algoritmo (siguiente) que presenta su ayuda **Suma. Guarda** este algoritmo en la carpeta **Pseudocódigo**. Muestra algoritmo para sumar dos números y mostrar el resultado, sirve como ejemplo para observar la estructura básica de un programa. Además de la sintaxis obligatoria, se incluyen comentarios que ayudan a entender el código, y se separan claramente las tres partes más usuales: carga de datos, cálculo, y presentación del resultado.

```
// Ejemplo simple de la ayuda de PseInt
// Algoritmo que toma dos números, los suma y muestra el resultado

Algoritmo Suma


   // para cargar un dato, se le muestra un mensaje al usuario
   // con la instrucción Escribir, y luego se lee el dato en
   // una variable (A para el primero, B para el segundo) con
   // la instrucción Leer

   Escribir "Ingrese el primer numero:"
   Leer A

   Escribir "Ingrese el segundo numero:"
   Leer B


   // ahora se calcula la suma y se guarda el resultado en la
   // variable C mediante la asignación (<-)

   C <- A+B


   // finalmente, se muestra el resultado, precedido de un
   // mensaje para avisar al usuario, todo en una sola
   // instrucción Escribir

   Escribir "El resultado es: ",C

FinAlgoritmo
```

2. Datos. Tipos de datos

Para que el ordenador realice el proceso automático, debe tener almacenada información en su interior.

La parte del ordenador en la que se almacena la información es la **Memoria Principal (RAM** =Memoria de acceso directo).

La memoria, está constituida por multitud de posiciones de memoria (celdas de memoria) numeradas de forma consecutiva, que almacenan tanto datos, valores como acciones (instrucciones).

Las **operaciones** que se pueden realizar en la memoria son la **Lectura** y la **Escritura**.

En la operación de **lectura**, se selecciona la posición de memoria y se obtiene el valor que contiene y en la **escritura**, se selecciona la posición de memoria en donde se desea escribir y el valor, de forma que se grabará dicho valor en esa posición; tanto si estaba libre como ocupada. Por eso, la operación de escritura en memoria es ***destructiva*** porque se pierde la información que hubiera en esa celda de memoria.

2.1. Tipos de datos

Los tipos de datos que se manejan en Pseudocódigo pueden ser *simples* o *estructurados*. Veremos, en primer lugar, los datos simples.

2.1.1.Tipos de datos Simples

- Numéricos (ENTERO o REAL)
- Lógicos
- Carácter
- Cadenas

Los tipos de datos numéricos **enteros** puedes ser positivos o negativos. Ejemplos de números enteros: -5, 8, -18, 1028, etc.

Los numéricos **reales** son valores numéricos con componente decimal (se utiliza el punto para indicar los decimales) y pueden ser también positivos o negativos. Por ejemplo: -0.8, 3.45, 7.8 etc.

Son de tipo **lógico** los valores que pueden ser: verdadero (1) y falso (0). Se utilizan en la determinación de las condiciones. Por ejemplo, cuando se indica una condición como: “8 es mayor que 4”, la respuesta es el valor verdadero (1).

Los valores de tipo **carácter**, son un conjunto finito de caracteres que el ordenador reconoce y que se compone por las letras del alfabeto en minúsculas y mayúsculas, cifras 0 al 9 y caracteres especiales. Todos estos caracteres puedes visualizarlos en la tabla ASCII.

Ejemplos de valores de tipo carácter: 'a ' , 'C', '8' , '#' etc. (van entre comillas simples).

Los **cadenas** es una estructura de datos de tipo carácter. Pero se le denomina cadena porque debe tener más de un carácter. La cadena debe ir encerrada entre comillas dobles. Ejemplo: “hola”.

Los *tipos de datos* se determinan *automáticamente* cuando se crean las variables. Por ejemplo, en la operación de ***asignación*** (como veremos más adelante) de la variable "***B <- 6;***" se está indicando implícitamente que la variable ***B*** es numérica.

3. Constantes y Variables

Para que el ordenador pueda utilizar valores durante la ejecución de un proceso, es necesario guardar dichos valores en posiciones de memoria del mismo. Estos valores pueden variar o no durante su ejecución. Así distinguimos entre: **Constantes** y **variables**.

Constantes: Son valores que no cambian durante la ejecución de un proceso. Se identifican por un nombre (normalmente en Mayúsculas) y su valor asociado. Ejemplo: VERDADERO = 1 , CIEN= 100, IVA = 0.21

Variables: Son valores que cambian durante la ejecución de un proceso. Se identifican por un ***nombre*** (que siempre debe comenzar por una letra) y el ***tipo de datos*** asociado que almacenará la variable. Siempre deben ser **Inicializadas** (normalmente a 0) Ejemplos: A, Total, Dirección, etc.

Hay dos formas de **crear una variable** y asignarle un valor. Son: la **Lectura** y la **asignación**.

Lectura
Desde el teclado, el usuario introduce un valor para esta variable.
Si la variable existe, ésta toma el valor introducido por el teclado
Si no existe la variable, se crea con ese valor.

Leer *Nombre*

Asignación
La operación o instrucción de asignación permite almacenar un valor en una variable.
Cuando se ejecuta la operación de asignación se evalúa el valor o expresión de la derecha y luego se asigna el resultado a la izquierda.

Nombre *< - "Andrés"*

ACTIVIDADES

Indica, en un fichero **Variables,** cuáles de las siguientes variables o constantes son correctas y cuáles no lo son. **Guarda** el fichero en la carpeta **Pseudocódigo**.

NOMBRE	TIPO DE DATOS	VALOR	RESPUESTA
Numero	carácter	"%"	
Nombre	cadena	"Fuente"	
NOMBRE	cadena	245.68	
NOMBRE	Num. Real	45.78	
VERDADERO	lógico	1	
NIF	Num. Entero	65.8	
NIF	Cadena	"9087654K"	
NUMPAG	Num. Real	"238"	

Numero Num. Entero -128

4. Operadores y expresiones

Son los elementos que permiten indicar la realización de una operación entre determinados valores. La **sintaxis** de estas expresiones con operadores será siempre de la siguiente forma:

Expresión1 ***operador*** ***Expresión2***

Debemos tener en cuenta que cada **expresión** puede ser un solo valor, constante o variable, o una expresión como conjunto de valores relacionados mediante operadores, por lo que es necesario que existan unas reglas que indiquen en que **orden** deben efectuarse las operaciones. La *relación de los operadores* más usuales es la siguiente:

- Operadores Aritméticos
- Operadores Lógicos
- Operadores Relacionales

Operador	Significado	Ejemplo
Aritméticos		
+	Suma	'abcd' + 'efgh'
-	Resta	B < - Ingresos - Gastos
*	Multiplicación	Importe< - Cant * Precio
/	División	a/b
^	Potencia	AreaC< - L^2
DIV	División Entera	Función ***Trunc*** (op1/op2)
MOD, %	Resto	Resto < - n1 MOD n2
Lógicos		
AND (& o **Y**)	Es VERDADERO si todos los valores son VERDADEROS	(8 > 4) & (6<3) Verdadero
OR (| o **O**)	Es VERDADERO si algún valor es VERDADERO	(4>2) | (3=7) Verdadero
NOT (**No** o ~)	Cambia al valor contrario	~(4 < 3) Falso
Relacionales		
>	Mayor que	A > B

>	Menor que	A < B
=	Igual	'abcd' = 'abcd'
>=	Mayor o igual que	'b' >='a'
<=	Menor o igual que	4<=8
<>	Distinto que	'ac' <> 'bc'

ACTIVIDADES

1. Obtener los resultados de las siguientes expresiones:

a) 7*(8-4)/2*5+4
b) 7*8-4/2*(5+4)
c) (“abc” + “de”) > “abcde”
d) 7>6 AND 5=5 OR 4<0
e) 2^3+6/3-4^2
f) (2^(3+6/3)-4)^2
g) NOT (5<6 OR 7=0) AND Falso
h) 5.25 + 8.5 / 5 - 3.2 * 7.25

5. Declaración o definición de Variables

Una variable debe definirse antes de ser utilizada por primera vez.

Declarar o definir una variable significa indicar el tipo de datos que almacenará la variable durante la ejecución del programa.

Si se intenta asignar a una variable ya definida un dato de un tipo de datos incorrecto se producirá un error en tiempo de ejecución.

Con el uso de la herramienta **PseInt** no es necesario la declaración de variables, sin embargo, nos acostumbraremos a declarar variables y constantes de cara a la codificación en un lenguaje de programación. En **PseInt** se puede **declarar explícitamente** una variable con la palabra clave **definir.**

definir **<var1> , <var2>, ..., <varN>** **como [REAL/ENTERO/LOGICO/CARACTER]**

Ejemplo: ***definir*** A ***como*** numerico***;***

Otra alternativa más informal sería:

Su **sintaxis**:

Declaración de una variable
<var> es [REAL/ENTERO/LOGICO/CARACTER/CADENA];

Declaración de varias variables
<var1>, <var2>, ...<varN > son [REAL/ENTERO/LOGICO/CARACTER/CADENA];

Ejemplos:

Nombre es **cadena**;
contador, *suma* son **entero**;

6. Asignación de Variables

Ya hemos visto que una variable está relacionada con posiciones de memoria que van a contener valores que cambiarán durante la ejecución del proceso, por tanto es necesario tener una ***acción*** que nos permita dar los valores adecuados a cada variable para obtener al final del proceso los resultados correctos. Esa acción u operación es la **Asignación**.

Por lo tanto, una **Asignación** es una operación que nos permite dar valor a una variable.

Debemos tener en cuenta el tipo de variable, para asignarle un valor. Así, una variable de tipo *numerico*, no podrá contener un valor de *'abc'* que es de *carácter*. La asignación es una operación destructiva, es decir, que si almacenamos un valor en una variable, el valor que tuviera anteriormente se perdería.

Su **sintaxis**:
variable **<-** valor o expresión;

Recordemos que: "*Cuando se ejecuta la operación de asignación se evalúa el valor o expresión de la derecha y luego se asigna el resultado a la izquierda*"

ACTIVIDADES

1. Indicar cuales serán los valores finales que contendrán las distintas variables después de los procesos siguientes:

a) A <- 8.5
B <- 0
A <- A*5
B <- B-A
A <- B

b) A <- 0
B <- 7+ A
C <-A/B
B <-7 MOD 3
A <-A+2
C <-A\B

c) L <- Verdadero
F <-L OR NOT (5<7)
A <- (4 MOD 3) \2 +5
B <- A+7
F <- F AND A < B AND L

3) Realizar el intercambio de los valores que contienen dos variables A y B sabiendo que se debe utilizar una variable auxiliar (intermedia) AUX.

4) Realizar lo mismo que en el ejercicio anterior, pero con tres variables; del modo siguiente:

* B toma el valor de A
* C toma el valor de B
* A toma el valor de C

También se usará una sola variable auxiliar AUX

5)Inicio
A <- 5
B <- 7
C <- 2
A <- A + B + C
B <- C / 2
A <- A / B + A ^ C
C <- A+(B-C)-B
Fin

a) ¿Qué valor contiene A después de la quinta linea?
b) ¿Qué valor contiene B después de la sexta linea?
c) ¿Qué valor contiene A después de la séptima linea?
d) ¿Qué valor contiene C después de la octava linea?

7. Operaciones de entrada y salida en Pseudocódigo

Veamos como podemos introducir o mostrar información.

Para ***solicitar información*** al usuario, se debe utilizar la operación ***Leer***.

Su **sintaxis**:

leer <variable>

Ejemplo:

leer *dato*

Para mostrar por pantalla un valor, se usará la operación ***Escribir***

Su **sintaxis**:

escribir <variable>

Ejemplo:

escribir dato

También podemos poner un texto tras la operación escribir. Este texto, deberá ir entre comillas. Ayuda al usuario para saber qué tipo de datos deberá introducir.

Ejemplo:

Guarda el siguiente algoritmo en la carpeta **Pseudocódigo**

```
Algoritmo entrada_salida

    nombre es cadena;
    escribir "Introduzca su nombre: ";
    leer nombre;

    escribir "Gracias" nombre;
finalgoritmo
```

8. Variables Especiales

- **Contadores**. Se utilizan para contar un evento que ocurra dentro del programa. Suelen contar desde 0 y de 1 en 1. Aunque pueden realizar otro tipo de cuenta, según el incremento aplicado. Se utilizan realizando dos operaciones:
    1. ***Inicialización***. Todo contador e inicializa a 0 si se va a realizar

una cuenta natural. O a un Valor Inicial (Vi) si se desea contar a partir de otro valor que no sea el 0.

CONTADOR < - 0

2. ***Incremento***. Cada vez que aparece el evento a contar se incrementa en 1 el contador o en otro valor.

CONTADOR < - CONTADOR + CONSTANTE

La ***constante***, puede ser un **1** u otro **valor**.

- **Acumuladores**. Se utilizan para acumular resultados parciales de cálculos con una misma operación. En general se utilizan para calcular sumas y productos. Se realiza en dos operaciones:
    1. **Inicialización**. El acumulador deberá ser inicializado con un valor neutro. Si la operación es una suma se inicializa en 0 y si es producto en 1.

SUMA <- 0
PRODUCTO <- 1

2. **Acumulación.** La acumulación se realiza por medio de las siguientes asignaciones, según la operación:

SUMA <- SUMA + VARIABLE
PRODUCTO <- PRODUCTO * VARIABLE

Ejercicio 1:
Realiza el pseudocódigo de un programa que permita escribir el resultado de la suma 2 números. Nombre del *Algoritmo suma_2* y **guárdalo** en la carpeta **Pseudocódigo**.

Objetos:

num1, num2	Variables que almacenan los datos del usuario
SUMA	**Acumulador** para calcular la suma

Solución:

```
Algoritmo suma_2
    //Ejercicio1. Calcula la suma de dos números
        //Declaración e inicialización de la variable suma
        suma <- 0;
        //se piden los datos al usuario y se almacenan en las variables num1, num2
        escribir "Introduzca el valor de dos números enteros:"
        leer num1, num2;
        suma <- num1 + num2;
        escribir "La suma de los números es:";
```

```
        escribir suma;
FinAlgoritmo
```

Ejercicio 2:
Algoritmo que calcule y escriba la suma y el producto de los 10 primeros números naturales. *Nombre contador_acumulador* y **guárdalo** en la carpeta **Pseudocódigo**.

Objetos:

I	**Contador** de 1 a 10
SUMA	**Acumulador** para calcular la suma
PRODUCTO	**Acumulador** para calcular el producto

Solución:

```
Algoritmo contador_acumulador
        //Uso de variables como contador y acumulador para sumar los 10
primeros números        //naturales
        //Declaración de las variables
        i,suma,producto son Entero;
        //Inicialización de las variables
        i <- 0;
        suma <- 0;
        producto  <- 0;
        mientras i <= 10 hacer
                suma <- suma + i;
                i <- i +1;
        FinMientras
        escribir "La suma de los 10 números naturales es:";
        escribir suma;
FinAlgoritmo
```

- **Interruptores o commutadores (switches).** Se utilizan para transmitir información de un punto a otro dentro del programa. Sólo pueden tomar dos valores:
-
    Numérico: 0 o 1
    Lógico: Verdadero o falso

Se inicializan en un valor y en los puntos que corresponda se cambian al valor contrario. Esto es de utilidad en las estructuras de control y los bucles.

Ejercicio1:
Algoritmo que lea una secuencia de notas (con valores de 0 a 10) que termina con el valor -1 y nos dice si hubo o no alguna nota con valor 10.

Objetos:
NOTA Variable para leer la secuencia.
SW **Interruptor** para controlar la aparición de una nota con los siguientes significados:
FALSO: No hay nota 10
VERDADERO: Si hay nota 10

Solución
Guarda el siguiente algoritmo en la carpeta **Pseudocódigo**.

```
Algoritmo variable_interruptor
    //Uso de una variable como Interruptor para leer notas. Ejercicio 1.
    //Declaración de las variables
    nota es entero;
    sw es logico;
    //Inicializar el interruptor sw
    sw <- falso

    //Leer desde el teclado una nota
    escribir "Introduzca una secuencia de notas (enteros). Para indicar el final pulse -1:";
    leer nota;
    mientras nota <> -1 hacer
        //escribir nota;
        si nota = 10 entonces
            sw <- verdadero
            escribir "Es una nota 10"
        FinSi
        leer nota;
    FinMientras
FinAlgoritmo
```

ACTIVIDADES

Ejercicio 1:
Algoritmo que sume independientemente los pares y los impares de los números comprendidos entre 1 y 10. Nombre del *algoritmo par_impares.* **Guárdalo** en la carpeta **Activiades**.
Objetos:
I Contador de 1 a 10 que genera la serie
PAR, IMP Acumuladores para calcular la suma de pares e impares respectivamente.
SW Interruptor o Conmutador significando:
FALSO: Par
VERDADERO: Impar

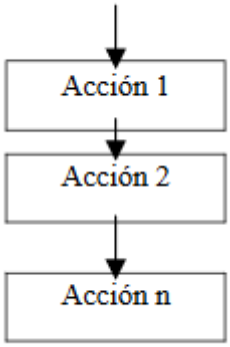
9. Estructuras de Control

Existen tres tipos de estructuras de control:

- Estructuras secuenciales
- Estructuras condicionales
- Estructuras repetitivas

9.1. Estructuras secuenciales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia.
La siguiente figura representa una estructura secuencial; como se aprecia este tipo de estructura tiene una entrada y una salida.



Ejemplo:
Pseudocódigo que calcula la suma y el producto de dos números. **Guarda** el algoritmo en la carpeta **Pseudocódigo**.

Solución:
Algoritmo suma_prod

```
//Algoritmo que calcula la suma y el producto de dos números introducidos por teclado
//Declaración de las variables
n1,n2,suma,producto son entero;
suma <- 0; producto <- 0;
escribir "Introduce dos números enteros: ";
leer n1,n2;
suma <- n1+n2;
producto <- n1*n2;
escribir "La suma y producto de los dos números enteros es: ";
escribir "Suma:" suma "," "Producto:" producto;
```

FinAlgoritmo

9.2. Estructuras de control condicionales o selectivas

Durante la especificación formal de un algoritmo, pueden presentarse problemas que exigen una descripción más complicada que la secuencial. Es el caso de que se requiera resolver un situación donde se dan varias alternativas tras la evaluación de una condición.

La forma de resolver esta situación es, utilizando las estructuras selectivas o condicionales. Que pueden ser:

- Condicional simple
- Condicional doble
- Condicional múltiple

En las estructuras condicionales o selectivas se evalúa una ***condición*** y en función del resultado de la misma se realiza una opción u otra. Las condiciones se especifican usando *expresiones lógicas* que deben dar un resultado **Verdadero** o **Falso** para ello, es habitual utilizar los operadores lógicos y relacionales.

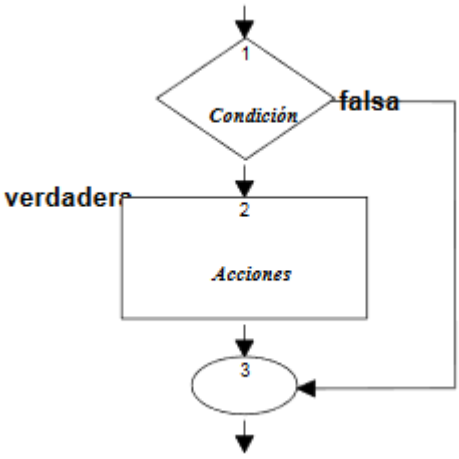
9.2.1. Condicional simple (*si-entonces*)

La estructura condicional simple (si-entonces) ejecuta una determinada acción cuando se cumple una condición. La condicional simple evalúa la condición y:

- Si la condición es verdadera entonces ejecuta la acción o acciones.
- Si la condición es falsa, no hace nada. La ejecución del programa continúa con la siguiente instrucción o acción secuencial.

Representación gráfica de la estructura condicional simple:

a) Ordinograma:



b) **Sintaxis** en Speudocódigo:

si <condición> **entonces**
acción 1 (o acciones)
fin_si

Ejemplo:
Algoritmo que comprueba si un número es distinto de positivo. Nombre *Algoritmo positivos*. **Guárdalo** en la carpeta Pseudocódigo.

Solución:

Algoritmo positivos
//Algorimo que comprueba que un número, introducido por teclado, es positivo.
//Declaración de variables

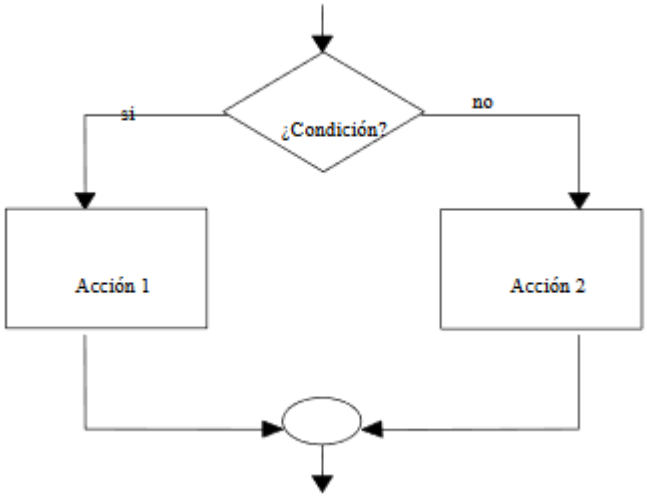
```
        numero es entero;
        escribir "Introduce un número entero positivo:";
        leer numero;
        si numero > 0 Entonces
                escribir  numero ", es un número positivo";
        FinSi
FinAlgoritmo
```

9.2.2. Condicional doble (si-entonces-si_no)

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre *dos opciones o alternativas posibles,* en función de si se cumple o no una determinada condición.

Si la condición es ***verdadera*** (***si***), se ejecuta la *acción 1* y, si es ***falsa*** (**no**), se ejecuta la *acción 2*

a) Organigrama:



b) ***Sintaxis*** en *Pseudocódigo*:

```
si <condición> entonces
        acción 1 (o acciones)
si-no
        acción 2 (o acciones)
fin-si
```

Ejemplo:
Algoritmo que resuelva una ecuación de primer grado. Nombre *Algoritmo Ecuación*. ***Guárdalo*** en la carpeta *Pseudocódigo*.

Si la ecuación es ax + b = 0; a y b son los datos. Las soluciones de la ecuación

son:

- Si a <> 0 x = -b/a Tiene solución
- Si a = 0 y b<>0 No tiene solución
- Si a = 0 y b = 0 Infinitas soluciones

Solución:

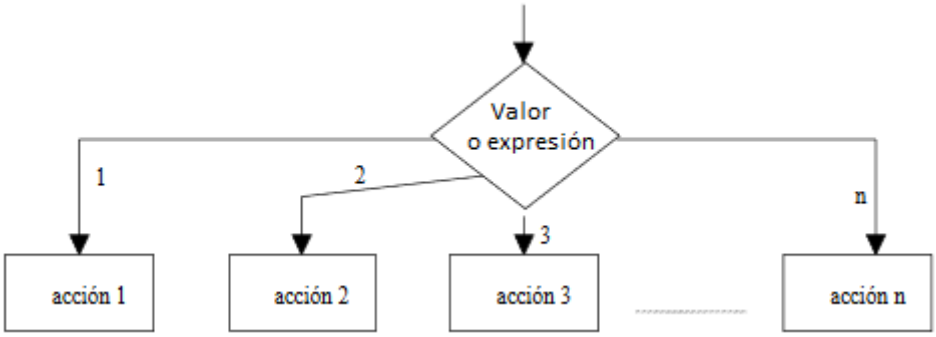
```
Algoritmo ecuación
	// Algoritmo que resuelve una ecuación de primer grado de la forma ax+b=0
	//Soluciones:
	//a<>0			Tiene solución x = -b/a
	//a=0,b<>0		No tiene solución x = -b/0
	//a=0, b=0		Tiene infinitas soluciones. Solución Indeterminada
	//Declaración de las variables
		a,b,x son reales;
	escribir "Se le pedirá los datos a,b para resolver la ecuación ax+b=0";
	escribir "Introduzca un valor para a:"
	leer a;
	escribir "Introduzca un valor para b:"
	leer b;
	si a <> 0 Entonces
		x <- -b/a
		escribir "La solución de la ecuación ax+b=0 es:";
		escribir "x=-b/a, x=" x;
	SiNo
		si b <> 0 Entonces
			escribir "No tiene solución. x=-b/0, x=" x;
		SiNo
			escribir "Tiene soluciones infinitas."
			escribir "Solución indeterminada. 0x=0";
		FinSi
	FinSi
FinAlgoritmo
```

9.2.3 Condicional múltiple (según <variable> hacer)

A veces, puede que sea necesario resolver un problema, en el que se planteen más de 2 elecciones posibles. El problema podría resolverse anidando varias de las estructuras anteriores. Sin embargo, eso presenta varios inconvenientes: puede ser complejo escribir el algoritmo y no será legible. Un solución es utilizar una estructura múltiple.

Una estructura de decisión múltiple evaluará una variable o una expresión, ***no puede ser una condición***, que podrá tomar **n valores distintos : 1, 2, 3, 4, ...n.** Según que elija uno de estos valores se realizará una de las **n acciones posibles.**

a) Ordinograma:



b) **Sintaxis** en Pseudocódigo:

Segun (valor o **expresion) hacer**
valor1: acciones
valor2: acciones
….
valorN: acciones
de otro modo: acciones
FinSegun

Ejemplo:
Se desea diseñar un algoritmo que escriba los nombres de los días de la semana en función del valor de una variable DIA introducida por teclado, que representa su posición dentro de la semana. **Guarda** el algoritmo en la carpeta **Pseudocódigo**.

Solución:

```
Algoritmo nombre_días
    //Algoritmo que muestra el nombre del día de la semana según su posición
    //introducida por teclado
    //Declaración de variables
    dia es numerico;
    Escribir "MOSTRAR EL NOMBRE DEL DÍA DE LA SEMANA";
    escribir "Introduce, con un número, la posición de un día de la semana:";
    leer dia;
    segun dia hacer
        1: escribir "El día de la semana es Lunes";
        2: escribir "El día de la semana es Martes";
        3: escribir "El día de la semana es Miércoles";
        4: escribir "El día de la semana es Jueves";
        5: escribir "El día de la semana es Viernes";
        6: escribir "El día de la semana es Sábado";
        7: escribir "El día de la semana es Domingo";
```

```
            de otro modo: escribir "ERROR";
        FinSegun
FinAlgoritmo
```

9.3. Estructuras Repetitivas o Bucles en Pseudocódigo

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan **bucles**, y se llama **iteración** al hecho de repetir la ejecución de una secuencia de acciones. Veamos un ejemplo:

Ejemplo:
Algoritmo que sume una ***lista de números*** introducidos desde el teclado, por ejemplo las notas de los alumnos de una clase. **Guarda** el algoritmo en la carpeta **Pseudocódigo**.

Solución:

```
Algoritmo sumar
        //Algoritmo que suma las notas de los alumnos introducidas desde el
teclado
        //Declaración de variables
        nota, suma es enteros;
        //Inicializar la variable suma (acumulador)
        suma <- 0;
        escribir "Introduzca el valor de una nota:";
        leer nota;
        suma <- suma + nota;
        leer nota;
        suma <- suma + nota;
        leer nota;
        suma <- suma + nota;
FinAlgoritmo
```

Vemos que por cada uno de los valores de la lista. El algoritmo repite muchas veces las mismas acciones:

leer nota;
suma < - suma + nota;

Estas acciones repetidas se denominan **Bucle** o **loop**. La acción (o acciones) que se repiten en un bucle, se denomina **iteracción.**

Cuando se utiliza un bucle para sumar una lista de números, se necesita saber cuantos números se han de sumar. Para ello necesitaremos conocer algún medio para detener el bucle.

Ejemplo 1:
Algoritmo suma-numeros que calcula la suma de **N** números que se introducen

por teclado. **Guarda** el algoritmo en la carpeta **Pseudocódigo**.
N representa el número de interacciones que se van a llevar a cabo. Podemos contar el número de interacciones de dos formas:

- Utilizando una variable **TOTAL** que se inicializa con el valor **N** y a continuación se decrementa cada vez que el bucle se repite.

TOTAL <- TOTAL -1;

- *Inicializando* la variable **TOTAL** en 0 o 1 e ir incrementando en uno en cada iteración del bucle hasta llegar a **N**.

Se puede resolver este algoritmo utilizando cualquiera de las siguientes estructuras de control repetitivas y que son:

- Estructura mientras
- Estructura repetir
- Para – Hasta

9.3.1. Estructura mientras

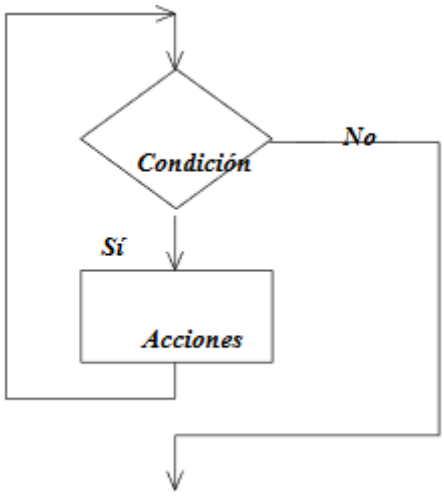
La estructura repetitiva **mientras** es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición.

Lo primero que se evalúa, en la instrucción mientras, es la condición (expresión booleana) antes de iniciar el bucle. Así:

- Si la condición es falsa no se ejecuta ninguna acción dentro el bucle, el programa continúa en la siguiente instrucción del bucle.
- Si la *condición* es ***verdadera*** se ejecuta el cuerpo del bucle y se vuelve a evaluar la condición en la siguiente iteracción.

a) Ordinograma:

:



b) **Sintaxis** en Pseudocódigo:

mientras condición **hacer**

```
            acción 1
            acción 2
            .....
            acción N
        fin_mientras
```

Solución utilizando la instrucción mientras <condición> hacer.

```
Algoritmo suma_numeros
        //Calcula la suma de los N números introducidos por teclado
        //Declaración de variables
N, TOTAL son entero;
NUMERO, SUMA son real;
escribir "Introduzca el número iteraciones que desea:";
        leer N;
        TOTAL <- N;      //Inicializamos la variable Total (contador) con el número de iteraciones que se van a llevar a cabo
        SUMA <- 0;       //Inicializamos la variable SUMA (acumulador) que almacenará la suma de esos números
        mientras  TOTAL > 0 hacer
                leer  NUMERO;
                SUMA <- SUMA + NUMERO;
                TOTAL <- TOTAL - 1;
        finmientras
                escribir "La suma de los ", N, " Números es " SUMA;
FinAlgoritmo
```

Cuando la variable TOTAL = 0, la condición se evalúa falsa y se sale del bucle mientras.

Ejemplo 2:
Algoritmo cuenta-enteros que cuente los números enteros positivos introducidos por teclado. Se consideran dos variables enteras NUMERO y CONTADOR (contará el número de enteros positivos). Se supone que se leen números positivos y se detiene el bucle cuando se lee un número negativo o cero. **Guarda** el algoritmo en la carpeta **Pseudocódigo.**

Ejercicio 1:

Modificar el anterior programa para que salga del bucle sólo cuando se le introduzca un cero. O sea, se podrán introducir números enteros positivos y negativos, pero sólo se contaran los positivos. **Guarda** el algoritmo con el nombre de **Algoritmo cuenta-enteros_2** en la carpeta **Actividades**.

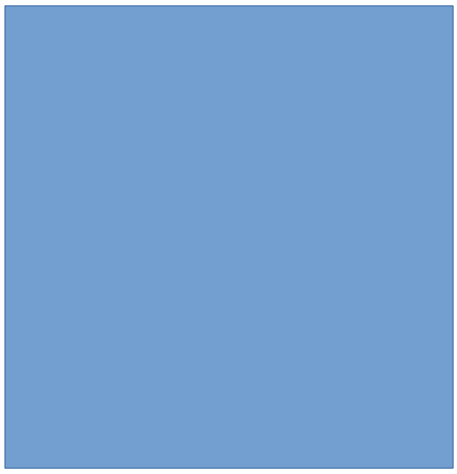
9.3.2. Estructura Repetir hasta-que

En algunas ocasiones el problema a resolver requiere que el bucle se ejecute una vez antes de

comprobar la condición de repetición. En la estructura **mientras** si el valor de la expresión booleana representada en la condición es falsa, el cuerpo del bucle no se ejecutará; por ello, se requiere que, al menos se ejecute una vez. La solución está en la existencia de otras estructuras repetitivas, como repetir y para – hacer, que sí lo hacen posible.
En la estructura **repetir** se evalúa la condición al final del bucle, por lo que las instrucciones del cuerpo del bucle, al menos, se ejecutan una vez.

El bucle **repetir-hasta_que** se repite mientras el valor de la expresión booleana de la *condición* sea **falsa**, justo la opuesta de la sentencia **mientras**.

a) Ordinograma: Realiza el organigrama correspondiente a esta variante en la sección ACTIVIDADES



b) Sintaxis de Pseudocódigo

Repetir
    acción 1;
    acción 2;
    …..
    acción N
hasta que *condición*

Este tipo de estructuras es típica para resolver situaciones en las que se solicitan datos de entrada al usuario y éstos han de filtrase. Es decir, deben comprobarse si se encuentran en un rango de valores establecidos.

También, esta estructura se utiliza para preguntar al usuario algo una vez y luego al final, preguntamos si desea continuar otra vez.

Ejemplo:
Algoritmo que pide al usuario que introduzca un número comprendido entre 2 y 8 En otro caso, no hace nada. **Guarda** el algoritmo con el nombre **Algoritmo uso_repetir** en la carpeta **Pseudocódigo**.

Solución

```
Algoritmo uso_repetir
                //Algoritmo que solicita al usuario que introduzca un número
                //comprendido entre 2 y 8
                //Declaración de variables.
                num es entero;
                escribir "Introduce un número comprendido entre 2 y 8.";
                escribir "El bucle se ejecutará una vez, aunque introduzcas otro
número distinto al rango.";
                escribir "En la segunda iteracción, ya no haría nada.";

                leer num;
                Repetir
                  escribir "Para continuar. Introduce otro número  comprendido
                  entre 2 y 8";
                  leer num;
                Hasta Que (num >= 2) o (num <=8)
FinAlgoritmo
```

ACTIVIDADES

Ejercicio1:
Diseña el ordinograma correspondiente a la estructura repetitiva o bucle **repetir-hasta que. Guárdalo** con el nombre **Ordinograma** en la carpeta **Actividades**.

Ejercicio 2:
Se deberá realizar un Algoritmo que detecte cualquier entrada comprendida entre 1 y 12, rechazando las restantes, ya que se trata de leer los números correspondientes a los meses del año. Deberá imprimir el nombre del mes correspondiente al número introducido por el usuario. **Guarda** el algoritmo con el nombre de **Algoritmo muestra_mes** en la carpeta **Actividades**.

9.3.2.1 Diferencias entre las estructuras mientras y repetir-hasta

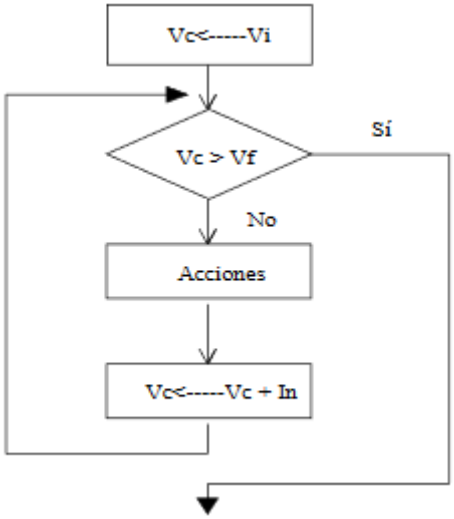
- La estructura **mientras** termina cuando la condición es **falsa**, mientras que **repetir** termina cuando la condición es **verdadera**.

- En la estructura **repetir** el cuerpo del bucle se ejecuta siempre **al menos una vez**; por el contrario, **mientras** permite la posibilidad de que el bucle pueda **no** ser ejecutado.

9.3.3. Estructura Para-hasta

Esta estructura ejecuta una secuencia de instrucciones un número determinado de veces y de forma automática controla el número de iteraciones a través del cuerpo del bucle.

a) Ordinograma:



b) **Sintaxis** en Pseudocódigo:

Para <variable(Vc)> < - <valor inicial(Vi)> **hasta** <valor final (Vf>
con paso constante **hacer**
acciones o instrucciones
Finpara

9.3.3.1 Diferencias entre las estructuras mientras y para-hasta

- La estructura **mientras** inicializa la variable fuera del bucle y la aumenta dentro del bucle.
- La estructura **para-hasta** inicializa e incrementa la variable en la misma línea de código.

El uso de una u otra depende del enunciado del problema. Así, cuando se conoce el valor del número de iteraciones, se usa **para-hasta**. Cuando no se conoce ese valor, entonces es mejor utilizar el bucle **mientras**. Por ejemplo, la suma de los números que introduzca el usuario, sabemos cuándo empieza pero no cuando acaba.

También debemos tener en cuenta que en la estructura o bucle **mientras** para que se ejecuten las acciones del cuerpo del mismo, la **condición** debe ser verdadera. En la estructura **para** debe ocurrir que el **Vc <= Vf** (si el incremento es positivo) o **Vc >= Vf** (si el incremento es negativo).

10. Funciones Matemáticas (Funciones Internas)

Para utilizar estas funciones se coloca su nombre seguido de los argumentos o parámetros **(x)** entre paréntesis. Por ejemplo **trunc(x)**. Se pueden utilizar como parámetros cualquier expresión, un número, una cadena o una variable. Cuando se evalúe la expresión, se reemplazará por el resultado correspondiente y se deberá almacenar en una **Variable**. La funciones también pueden usarse en una

condición de una estructura de control

FUNCIÓN	SIGNIFICADO
RAIZ(X)	Raíz cuadrada de X
ABS(X)	Valor absoluto de X
LN(X)	Logaritmo natural de X
EXP(X)	Función exponencial de X
SEN(X)	Seno de X
COS(X)	Coseno de X
TAN(X)	Tangente de X
TRUNC(X)	Parte entera de X
REDOND(X)	Entero más cercano a X
AZAR (X)	Entero aleatorio entre 0 y 1
LONGITUD(S)	Cantidad de caracteres de la cadena S
MAYUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en mayúsculas
MINUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en minúsculas
SUBCADENA(S,X,Y)	Retorna una nueva cadena que consiste en la parte de la cadena S que va desde la posición X hasta la posición Y. Empieza desde 0 y no desde 1.
CONCATENAR(S1,S2)	Retorna una nueva cadena resulta de unir las cadenas S1 y S2.

La función **raíz cuadrada (raiz(x))** no debe recibir un argumento negativo.
La función **exponencial (exp (x))** no debe recibir un argumento menor o igual a cero.

Recuerda que debes almacenar el resultado de las funciones en una variable. También, puedes usarla en una condición.

11. Análisis de un problema en pseudocódigo

Hasta ahora hemos visto cómo debemos construir el algoritmo en pseudocódigo, pero al tener un problema lo primero que debemos hacer es analizar el problema para saber cómo debemos construir el algoritmo.

Para empezar a analizar el problema, debemos leer atentamente el problema y extraer lo más importante.

Para extraer lo más importante podemos preguntarnos lo siguiente:

- **¿Debe el usuario introducir datos?**
- **¿Hay algún tipo de repetición dentro del algoritmo?**
- **¿Hay alguna estructura condicional?**
- **¿Qué condición sería la más adecuada, si tenemos un bucle o una estructura Si?**
- **¿Qué variables necesito en el algoritmo?**
- **¿Cómo debería iniciar las variables?**
- **¿Qué resultado debo imprimir en pantalla? (Si es que hace falta)**
- ***Ejemplos***:

  1 Un vendedor recibe un sueldo base más un 10 % extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que ha realiza este mes, y el total que recibirá.

  Empecemos analizando este enunciado, haciéndonos las preguntas vistas:

- **¿Debe el usuario introducir datos?** El usuario solo deberá introducir el valor de sus ventas
- **¿Hay algún tipo de repetición dentro del algoritmo?** En este caso, se podría hacer con o sin repetición pero siempre que se pueda es bueno hacer un bucle. Como en este caso sabemos el rango, podemos usar una estructura desde-hasta, aunque también se puede hacer con un mientas. Esto ya es decisión nuestra.
- **¿Hay alguna estructura condicional?** En este ejercicio no tenemos ninguna condición ya que nos pide que cumpla alguna condición concreta.
- **¿Qué condición sería la más adecuada, si tenemos un bucle o una estructura Si?** Como obtenemos del enunciado, nos pide 3 ventas, así que una buena condición seria que hasta una variable del tipo contador sea mayor que 3 (incluyendo el 3) no nos pida más ventas.
- **¿Qué variables necesito en el algoritmo?** Necesitaremos una variable que almacene el valor de la venta, una variable contador para contar el número de ventas, el total, el salario base y la comisión (estas dos últimas pueden ser constantes

- **¿Cómo debería iniciar las variables?** Las constantes se deberán asignar los valores que necesitemos, el total se inicializará a 0 y el contador a 1.

- **¿Qué resultado debo imprimir en pantalla? (Si es que hace falta)** Deberemos imprimir el total que recibirá el vendedor.

2. Escribe un algoritmo que determine cuáles son los múltiplos de 5 comprendidos entre 1 y N.

- Haremos lo mismo que antes, analizaremos el enunciado con las preguntas:

- **¿Debe el usuario introducir datos?** El único dato que debe introducir el usuario es el límite. Normalmente, cuando se dice **N** es un dato que debe introducir el usuario.

- **¿Hay algún tipo de repetición dentro del algoritmo?** Si tenemos una repetición, tendremos que ver si cada número es o no múltiplo de 5. Ahí podemos usar una estructura desde-hasta, ya que sabemos el rango (aunque sea un dato introducido por el usuario).

- **¿Hay alguna estructura condicional?** En este ejercicio si tenemos una estructura condicional, ya que imprimiremos solo aquellos números que sean múltiplos de 5.

- **¿Qué condición sería la más adecuada, si tenemos un bucle o una estructura Si?** En el caso del bucle, la condición para salir deberá ser cuando el número sea mayor que el límite. En el caso de la estructura condicional, la condición para que imprima el número es que sea múltiplo de 5.

- **¿Qué variables necesito en el algoritmo?** Simplemente necesitamos almacenar el límite y contar los números entre 1 y el límite.

- **¿Cómo debería iniciar las variables?** El variable contador se inicializará en uno.

El seguimiento es una técnica en la que comprobamos que el programa hace lo que queremos que haga, se puede hacer en papel o en el **PSeInt**.

En papel, lo único que se hace es comprobar el valor de la variables en cada línea y realizar la operaciones que aparezcan como leer, escribir, etc.

Inicio
contador<-1
suma<-0
Mientras contador<=2 **Hacer**
suma<-suma+contador

contador<-contador+1
FinMientras
Escribir suma
Fin

Esto es lo que iremos haciendo durante el seguimiento:

- Anotamos todas las variables que hay, en este caso, solo están las variables **contador** y **suma, contador** esta inicializado en 1 y **suma** en 0.

- Tenemos una estructura mientras con una condición **contador<=2,** ahora **contador** vale 1, **1<=2** es verdadero, por lo que entra en el bucle.

- Ahora realizamos la siguiente línea, **suma<-suma+contador**, es decir, **suma<-0+1,** es igual a **1.** Anotamos en **suma 1.**

- En la siguiente línea, **contador<-contador+1,** es decir, **contador<-1+1, contador** ahora vale **2**, lo reemplazamos por el valor que tenía.

- Volvemos al inicio del mientras, ahora **contador** vale **2**, la condición se sigue cumpliendo, **2<=2.**

- Se ejecuta las siguientes instrucciones como antes, pero con valores diferentes. La variable **suma** valdrá **3** y **contador 3**.

- Ahora en el mientras no se cumple la condición, ya que **3<=2** no es verdadero.

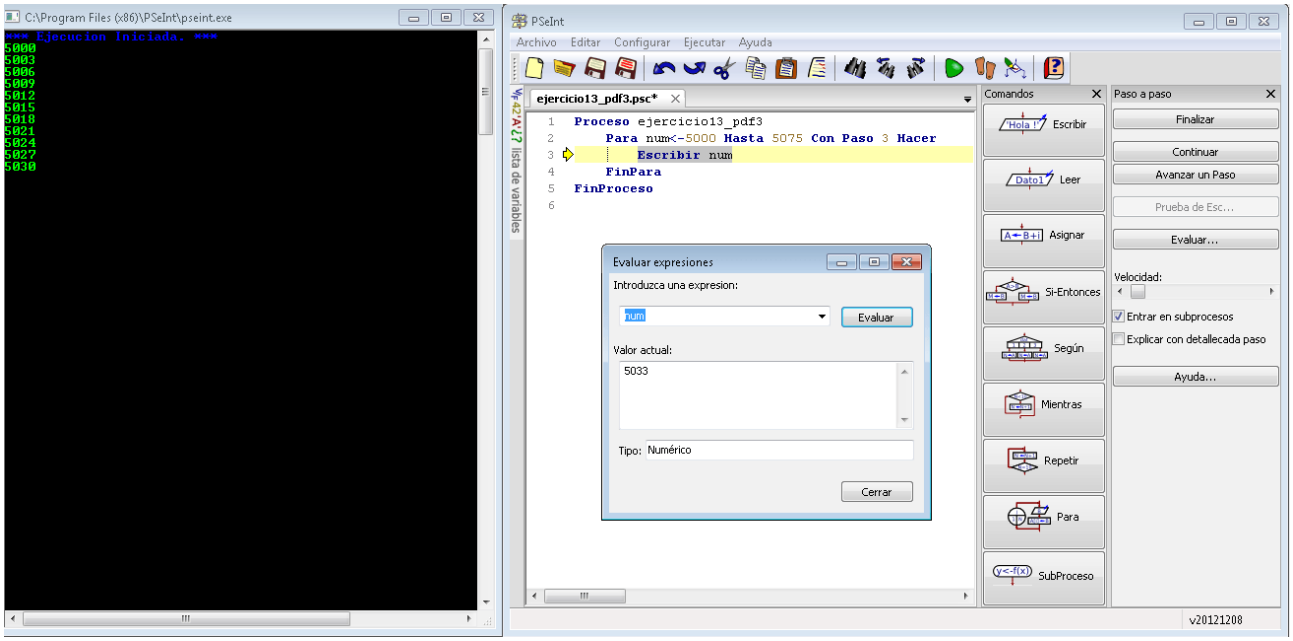
- Seguimos las instrucciones después del bucle, realizamos la línea **Escribir suma,** como **suma** vale **3**, lo escribiremos en pantalla.

Lo más aconsejable es comprobar los valores limites (en este caso 2) para ver que todo está bien. Si pedimos datos por teclado al usuario, debemos probar con varios números para ver que cumple lo que necesitamos.

En **PSeInt**, debemos pinchar en **"ejecución paso a paso"**, después en **"comenzar"**, aparecerá la ventana de ejecución e ira ejecutando línea a línea.

Si queremos ver el valor de una variable en un momento concreto, pausamos la ejecución y pinchamos en **"evaluar"** y ponemos el nombre de la variable que queremos ver.

También podemos ejecutarlo línea a línea y modificar la velocidad de ejecución.


C:\Program Files (x86)\PSeInt\pseint.exe
*** Ejecucion Iniciada. ***
5000
5003
5006
5009
5012
5015
5018
5021
5024
5027
5030
PSeInt
Archivo Editar Configurar Ejecutar Ayuda
ejercicio13_pdf3.psc*
1 Proceso ejercicio13_pdf3
2 Para num<-5000 Hasta 5075 Con Paso 3 Hacer
3 Escribir num
4 FinPara
5 FinProceso
6
Evaluar expresiones
Introduzca una expresion:
num
Evaluar
Valor actual:
5033
Tipo: Numérico
Cerrar
Comandos
Escribir
Leer
Asignar
Si-Entonces
Según
Mientras
Repetir
Para
SubProceso
Paso a paso
Finalizar
Continuar
Avanzar un Paso
Prueba de Esc...
Evaluar...
Velocidad:
Entrar en subprocesos
Explicar con detallecada paso
Ayuda...
v20121208